Machine Learning 1

Lecture 8.5 - Supervised Learning
Neural Networks

Erik Bekkers

(Bishop 5.3)

Slide credits: Rianne van den Berg

Image credit: Kirillm | Getty Images

Multi-dimensional chain rule

Recall "$NN(\underline{x}) = h \circ a^{(2)} \circ h \circ a^{(1)}(\underline{x})$"

Let $f : \mathbb{R}^D \mapsto \mathbb{R}$ be a differentiable function of D variables.

$a_1^{(\ell)}(a_1^{(\ell-1)}(\underline{x}), a_2^{(\ell-1)}(\underline{x}), \ldots)$ is a function of previous activations

Let $g_1, \ldots, g_D : \mathbb{R} \mapsto \mathbb{R}$ be differentiable functions, the inputs of f :

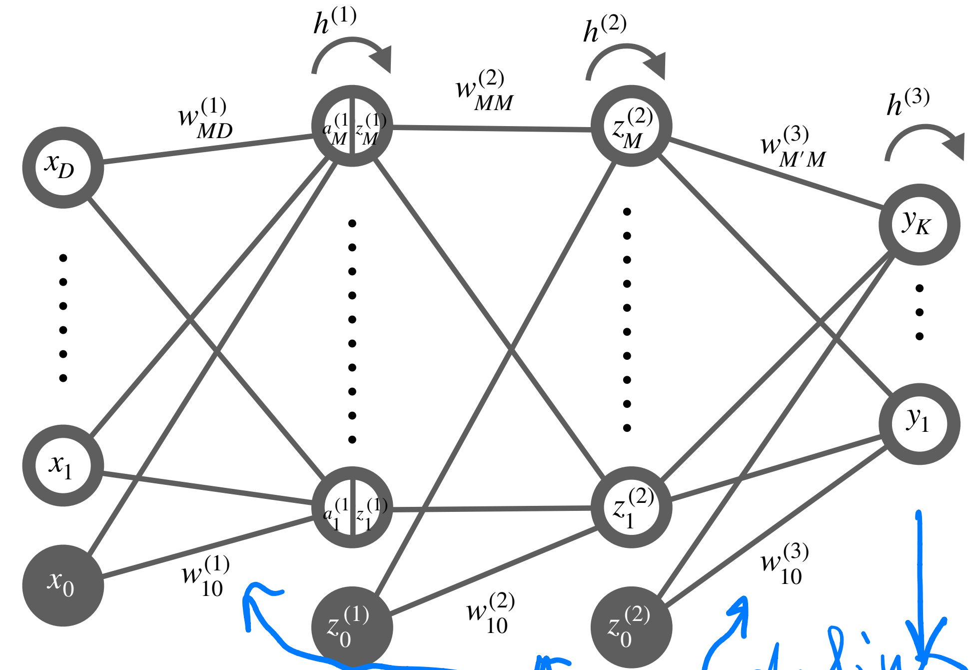
$a_1^{(\ell)}$ $a_d^{(\ell-1)}$

$$(g_1(x), \ldots, g_D(x)) \mapsto f(g_1(x), \ldots, g_D(x))$$

Then the **multi-dimensional chain rule** tells us the derivative to x is

$$\frac{\partial f(g_1(x), \ldots, g_D(x))}{\partial x} = \sum_{d=1}^{D} \frac{\partial f(g_1(x), \ldots, g_D(x))}{\partial g_d(x)} \frac{\partial g_d(x)}{\partial x}$$

Neural Nets: Gradient of Error functions



- Use $E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$

Objective

- Evaluate $\dfrac{\partial E_n(\mathbf{w})}{\partial \mathbf{w}}$

- For general feed-forward network:
  - Output/hidden activations: $a_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)}$
  - Output/hidden units: $z_j^{(l)} = h^{(l)}(a_j^{(l)})$

(defines Error

- Forward propagation: Compute all a_j and z_j

- Back propagation: Compute all derivatives $\dfrac{\partial E_n}{\partial w_{ji}^{(l)}}$

Neural Nets: Gradient of Error functions

- Back propagation: Compute all derivatives $\dfrac{\partial E_n}{\partial w_{ji}^{(l)}}$ corresponding to input $\mathbf{x}_n$

- E_n only depends on w_{ji} through activation: $\boxed{a_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)}}$

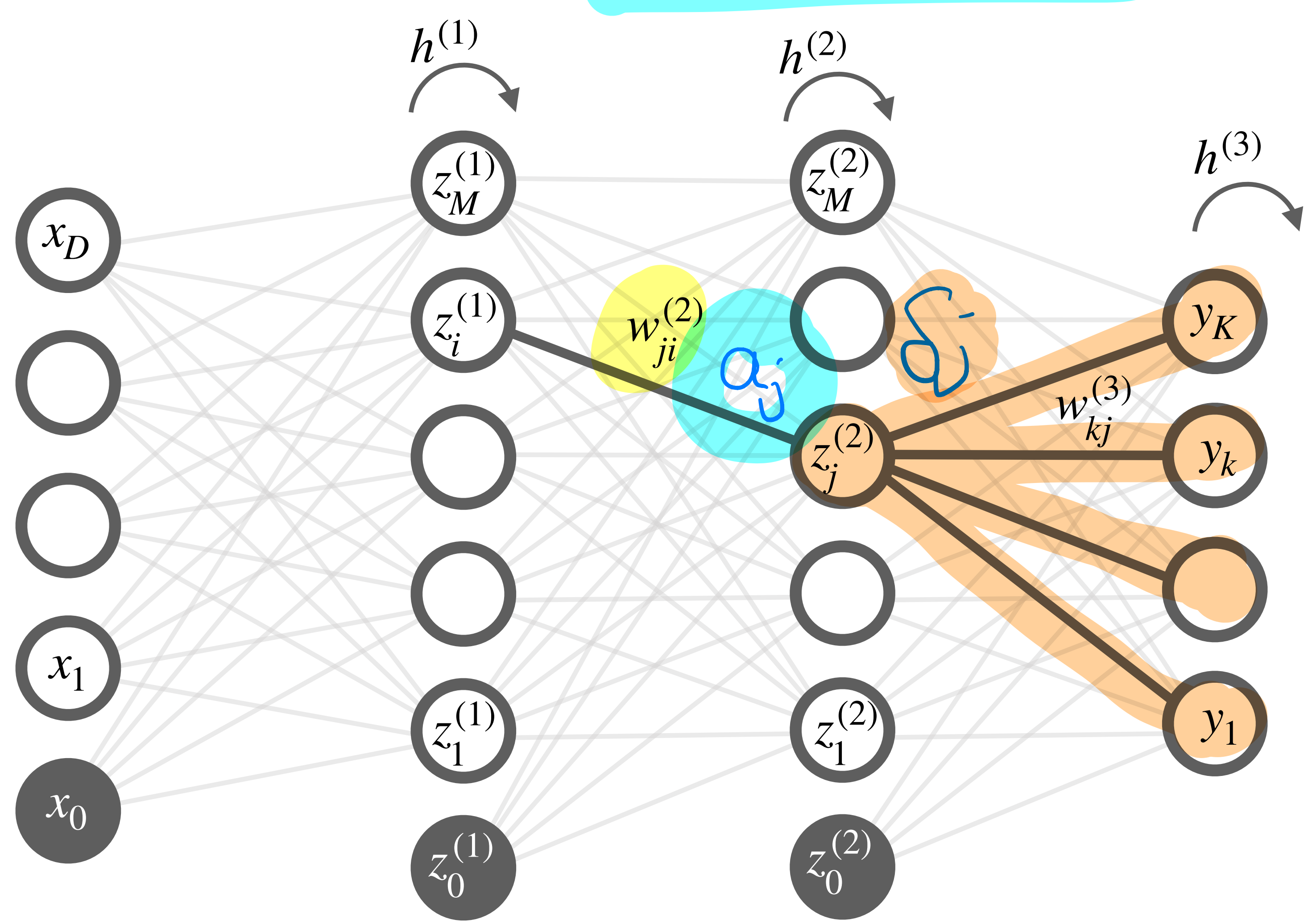
$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \frac{\partial E_n}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}$$

- Define "node error"

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

- Then

$$\frac{\partial E}{\partial w_{ji}} = \delta_j \cdot \frac{\partial a_j}{\partial w_{ji}}$$



Neural Nets: Gradient of Error functions

- Back propagation: - First compute all δ_j

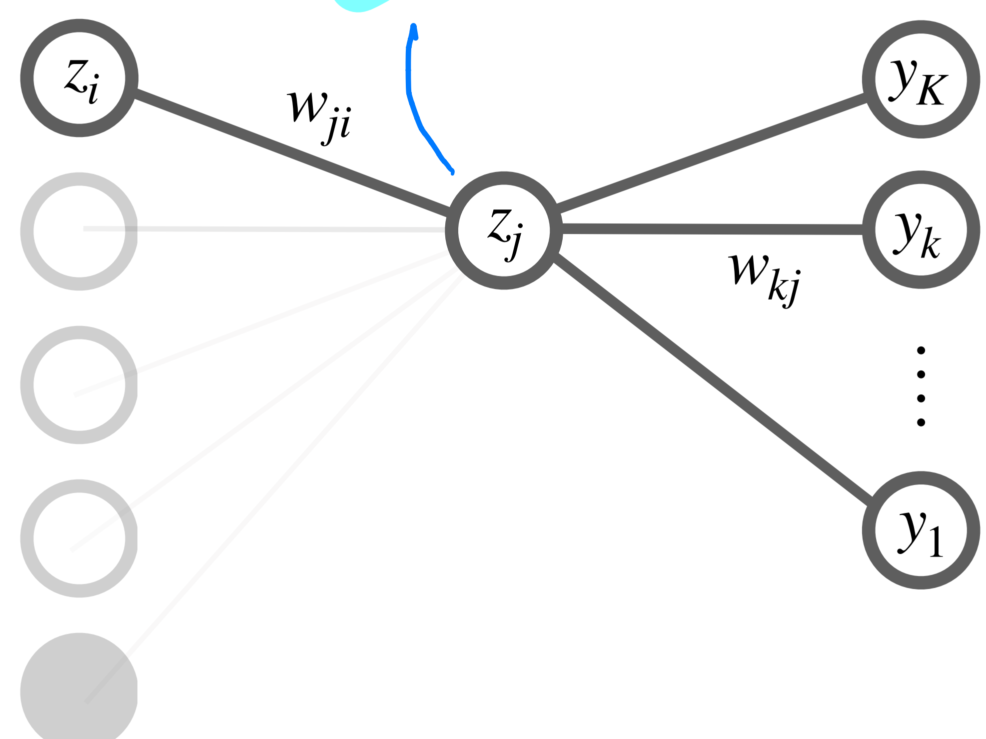
  - Then update all derivatives $\frac{\partial E_n}{\partial w_{ji}}$ $= \delta_j \frac{\partial a_j}{\partial w_{ji}}$

- We now omit layer indices and identify the layers with the indices i, j, and k

$$z_i^{(l-2)} = z_i$$
$$z_j^{(l-1)} = z_j$$
$$z_k^{(l)} = z_k$$

$i : (l-2)$ $j : (l-1)$ $k : (l)$

$a_j = \sum_i w_{ji} z_i$



- Now let's compute

$$\frac{\partial a_j}{\partial w_{ji}} = z_i$$

Neural Nets: Gradient of Error functions

- Back propagation: - First compute all δ_j

  - Then update all derivatives $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

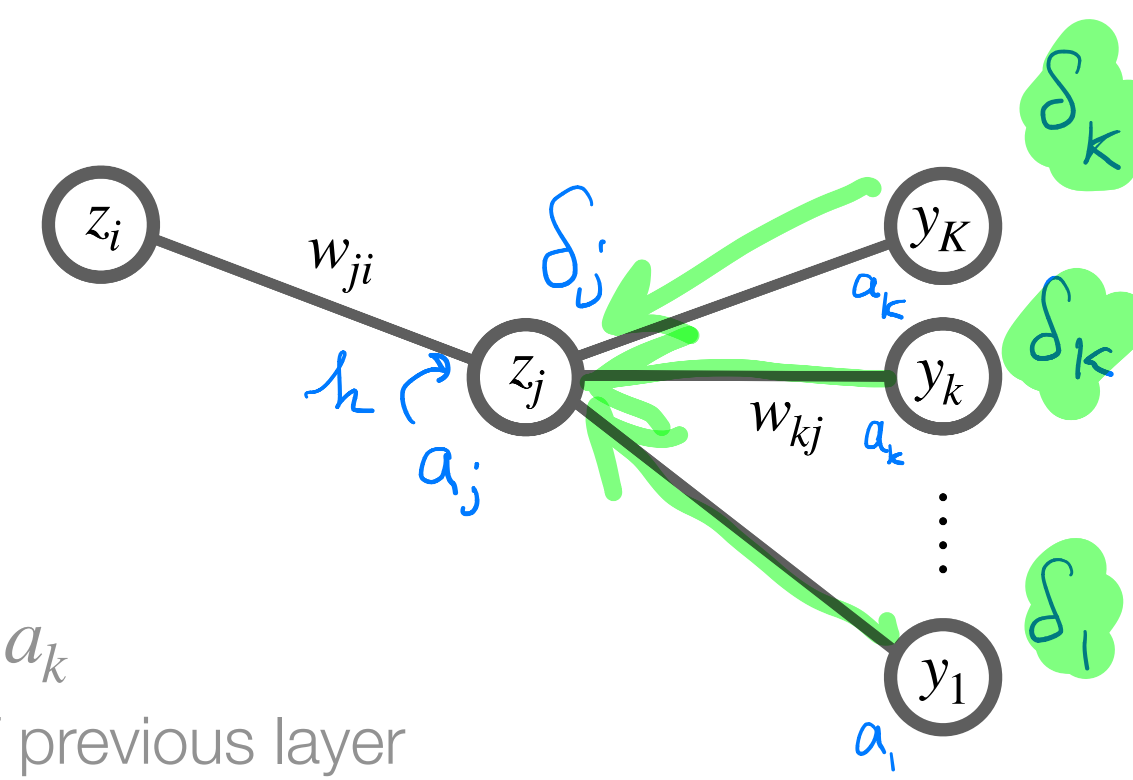
- We now omit layer indices and identify the layers with the indices i, j, and k

$E_n(\ldots, a_k(\ldots, a_j, \ldots), \ldots)$

- Now let's compute

$$\delta_j \equiv \frac{\partial E}{\partial a_j} = \sum_k \underbrace{\frac{\partial E}{\partial a_k}}_{\delta_k} \cdot \frac{\partial a_k}{\partial a_j}$$

$$= \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$



$E(a_1, \ldots, a_K)$ depends on output activations a_k

Outputs $a_k(a_1, \ldots, a_J)$ in turn depend on a_j of previous layer

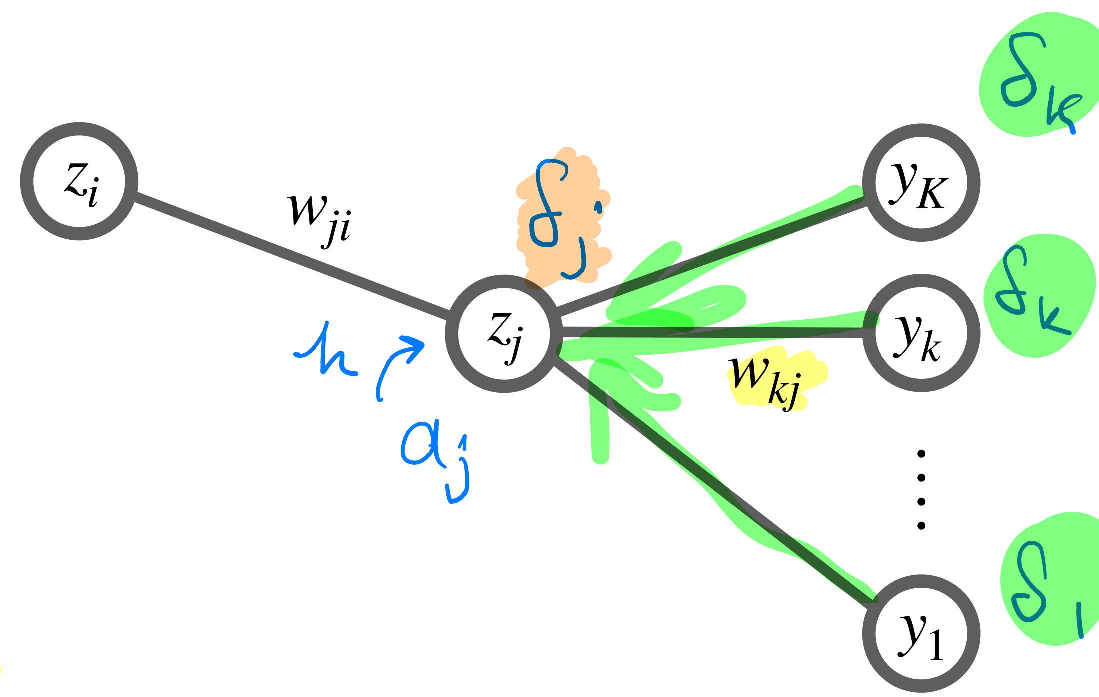
Use multi-dimensional chain rule!

Neural Nets: Gradient of Error functions

- Back propagation: - First compute all δ_j
  - Then update all derivatives $\dfrac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$
- We now omit layer indices and identify the layers with the indices i, j, and k
- So $\delta_j = \sum_k \delta_k \dfrac{\partial a_k}{\partial a_j}$, then let's compute $\dfrac{\partial a_k}{\partial a_j}$ (Recall: $a_k = \sum_j z_j w_k j$)

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial}{\partial a_j}\left(\sum_j w_{kj}\, h(a_j)\right)$$

$$= w_{kj}\, h'(a_j)$$



- Thus

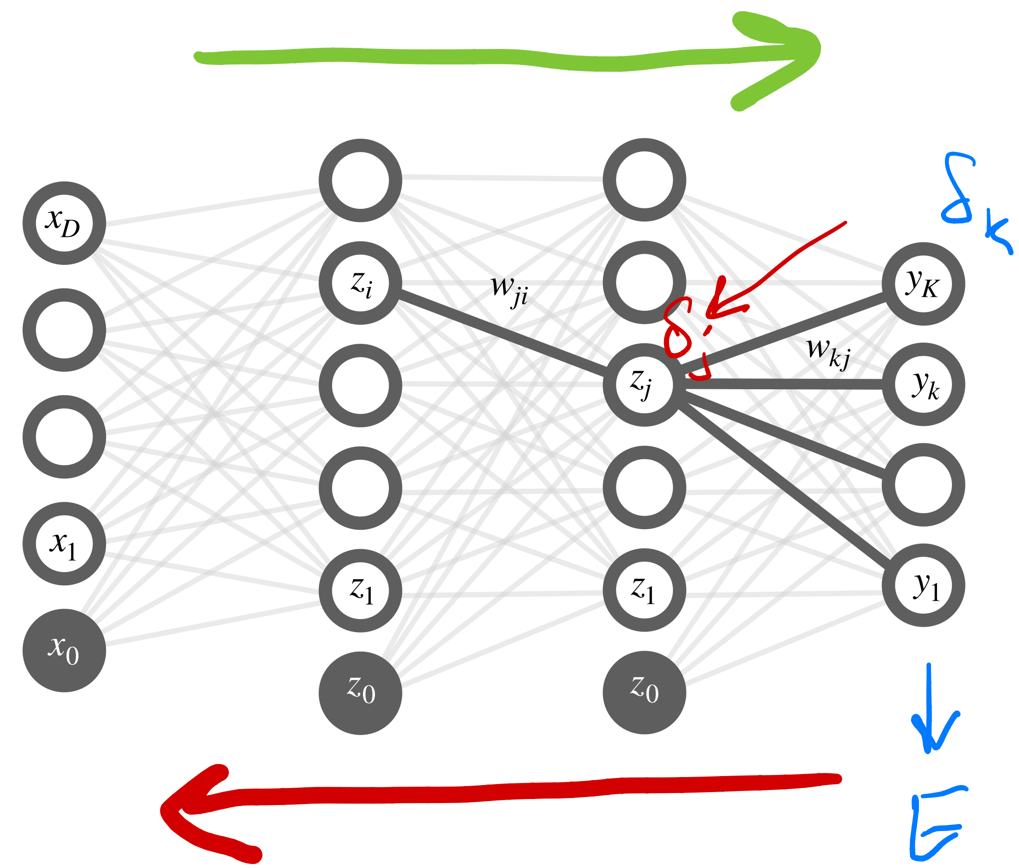
$$\delta_j = h'(a_j) \sum_k \delta_k w_{kj}$$

Forward and Backward Propagation

Forward propagation:

- For input $\mathbf{x}_n$ compute all hidden and output activations a_k and units z_k.

$$a_j = \sum_i w_{ji}\, h(a_i)$$



$$E = \frac{1}{2}(y_k - t_k)^2$$

Backward propagation:

- Compute δ_k for all output units.

$$\delta_k = \frac{\partial E}{\partial y_k} = (y_k - t_k)$$

- Compute δ_j for all hidden units through back-prop

(Careful with skip connections!)

$$\delta_j = h'(a_j) \sum_k w_{kj}\, \delta_k$$

- Compute derivatives

$$\frac{\partial E}{\partial w_{ji}} = \delta_j z_i$$

In general in any feed forward network where O_i denotes the set of (outgoing) node connections to node i, and w_{ni} the corresponding weights:

$$\delta_i = h'(a_i) \sum_{n \in O_i} \delta_n w_{ni}$$

Iterative weight updates:

$$w_{ji}^{(\tau+1)} = w_{ji}^{(\tau)} - \eta\, \delta_j z_i$$

Starting the backpropagation

- For regression: $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$

$$y(\mathbf{x}_n, \mathbf{w}) = y_n = a^{out}$$

$$\delta^{\text{out}} = \frac{\partial E_n}{\partial a^{\text{out}}} = y_n - t_n$$

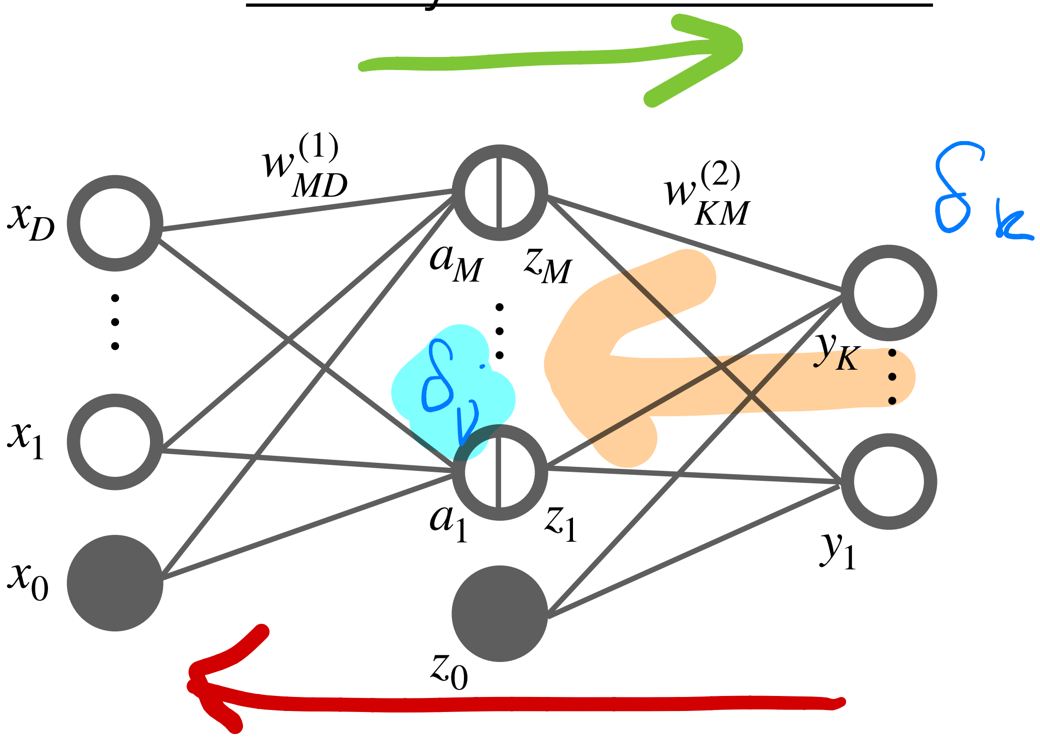
- For classification with K classes: $E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w})$

$$y_k(\mathbf{x}_n, \mathbf{w}) = y_{kn} = \frac{\exp(a_k^{\text{out}})}{\sum_{j=1}^{K} \exp(a_j^{\text{out}})}$$

$$\delta_k^{\text{out}} = \frac{\partial E_n}{\partial a_k^{\text{out}}} = y_{kn} - t_{kn}$$

Example: Backpropagation with tanh

- Two layer neural network:



- Regression with K outputs: $y_k = a^{(2)}_k$
- Hidden units: $z_j = h(a_j) = \tanh(a^{(1)}_j)$
- Activation function $h(a) = \tanh(a) = \dfrac{e^a - e^{-a}}{e^a + e^{-a}}$
- Has derivative $h'(a) = 1 - h(a)^2$
- Error function $E = \sum_k (y_k - t_k)^2$

- After forward propagation, compute:

$$\delta^{out}_k = y_k - t_k$$

- Backpropagate using:

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w^{(2)}_{kj} \delta^{out}_k$$

Update weights in first and second layer using:

$$\frac{\partial E_n}{\partial w^{(1)}_{ji}} = \delta_j x_i \quad \text{and} \quad \frac{\partial E_n}{\partial w^{(2)}_{kj}} = \delta^{out}_k z_j$$